

Developing a Video Steganography Toolkit

James Ridgway & Mike Stannett

Department of Computer Science, University of Sheffield,
Regent Court, 211 Portobello, Sheffield S1 4DP, United Kingdom

james@james-ridgway.co.uk m.stannett@sheffield.ac.uk

September 18, 2014

Abstract

Although techniques for separate image and audio steganography are widely known, relatively little has been described concerning the hiding of information within video streams (“video steganography”). In this paper we review the current state of the art in this field, and describe the key issues we have encountered in developing a practical video steganography system. A supporting video is also available online at <http://www.youtube.com/watch?v=Yhn1HmZolRM>.

Keywords. Steganography; video manipulation; covert communications; information hiding; tool development.

1 Introduction

This paper reports the findings of a seven-month dissertation project carried out at Sheffield University, investigating steganographic techniques for hiding data in video files encoded using the popular H.264 format (Ridgway, 2013). During the course of the project several key tools were developed, and these are described below together with experimental findings. All of the materials developed for the project can be accessed online at <http://www.steganosaur.us>. A supporting video is also available online at <http://www.youtube.com/watch?v=Yhn1HmZolRM>.

We begin by reviewing various existing approaches to digital steganography, before explaining in section 2 the specific issues that need to be addressed when developing these techniques for video container files. In section 3, we describe our experimental findings, and highlight areas where further research and development might be beneficial.

1.1 Background

Whereas encryption seeks to make a message uninterpretable to unauthorised eavesdroppers, steganography attempts instead to make the very existence of the message unsuspected (the two techniques can of course be combined; see section 2.4). Steganography has a very long history: Herodotus explains how Histiaëus, the ‘tyrant’ of Miletus, who was then staying with his overlord (the Persian emperor, Darius I), had a message tattooed onto a slave’s shaved head some 2500 years ago (499 BCE). Once the hair had grown back, the slave was sent to

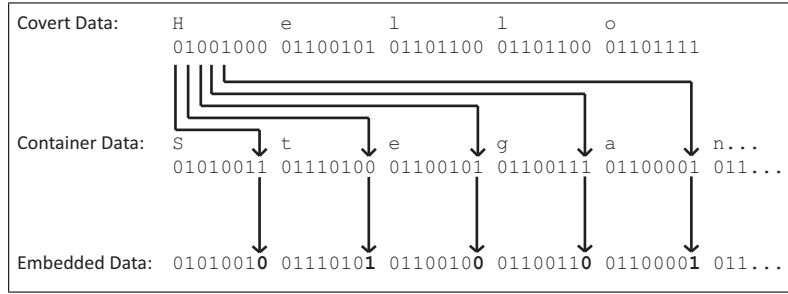


Figure 1: LSB encoding of the word “Hello” inside an arbitrary container file.

Miletus, where his nephew re-shaved the slave to find an instruction telling him to revolt against Darius (Marincola, 1996).

This idea, of hiding information covertly so that its presence is unsuspected even by eavesdroppers with access to the manipulated ‘container’ (in this case, the slave), can of course be adapted to modern digital communications. Perhaps the simplest approach to digital steganography is to *inject* data into redundant sections of a file. For example, because EXE files use an *end of file* (EOF) marker, adding additional data to the end of the file doesn’t affect executable behaviour. Other file types, e.g. WAV files, specify their intended size in a header (IBM and Microsoft, 2004), and additional data is again ignored. Although easy to implement, such injection techniques are extremely insecure, since direct analysis of the file can easily reveal the presence of unwarranted additional data.

In contrast, *substitution* techniques embed data in those sections of the file that are – relative to some appropriate metric – of least relevance, without affecting overall file size. This avoids, e.g., the tell-tail size inflation associated with injection techniques, but the *steganographic capacity* of the container is limited by the amount of ‘irrelevant’ data present. A particularly common substitution technique for audio and image files is *LSB manipulation* (Johnson et al., 2003; Cole, 2003; Fridrich, 2010) (Fig. 1), in which the least significant bit of each byte of an image, say, is manipulated so as to embed information without discernably changing the image as viewed on-screen. However, such techniques are inherently at odds with the lossy compression algorithms used by various digital encoding formats, since these specifically seek to disregard the same ‘irrelevant’ segments of a file, i.e., they ‘throw away’ precisely the segments where we want to hide our message. *Transform-domain* techniques can sometimes overcome this problem (section 1.4).

1.2 Video Steganography in the Literature

From a steganographic standpoint, video files have distinct advantages over stand-alone audio or image containers. In the first place, video files are typically much larger than other container files, and have far greater steganographic capacity. But in addition, video modification is also significantly harder for humans to detect than stand-alone image manipulation, because each video frame is only visible for a fraction of a second during normal playback, and moreover, video frames rarely include sharply focused images (Al-Frajat et al., 2010).

Surprisingly, however, the number of articles addressing video steganography appears to be rather limited, and those that exist in the literature generally give high level descriptions with only limited lower-level detail. Of those authors who specifically address the topic,

Node et al. (2004) suggest a *bit plane complexity segmentation* technique that hides data in wavelet-compressed video, and Jalab et al. (2009) give a related method for embedding data in MPEG video. Eltahir et al. (2009) discuss LSB manipulation of video, but do not address security. Finally, Singh and Agarwal (2010) specifically address the hiding of an image in a video, and while their method centres around LSB manipulation, this is one of the few papers that exploits the multi-dimensional aspect of a video as a container file; they claim, moreover, that the proposed technique is “very useful in sending sensitive information securely” but unfortunately they provide little supporting evidence for this claim, or for the effectiveness of the proposed technique.

1.3 Generation Techniques

Generation techniques involve generating a bespoke file from scratch by exploiting shared characteristics of the agents involved. For example, if Alice and Bob are both car enthusiasts, it is unlikely that the exchange of images showing pictures of the latest models will arouse much suspicion. We could therefore exchange a message covertly by creating an image of a race (say) in which the positions of the cars, spectators, or any other agreed component are used to encode the required information. Doing so may, of course, be time consuming, but this need not be an issue unless the information needs to be encoded and transmitted in real time. Such techniques have, moreover, a unique advantage over other steganographic methods, in that there is no underlying container file against which the transmitted file can be compared for steganalytic purposes. Reported research into generation techniques is, however, extremely limited, perhaps because the message construction process, with its heavy dependence on the shared interests of the specific agents involved, is necessarily ad hoc.

1.4 Transform Techniques

Discrete cosine transform (DCT) techniques are often used with compressed image files, and these can be applied, to some extent, to individual images within certain video streams provided the frames to be manipulated are chosen appropriately. Informally, the discrete cosine transform takes image descriptions given in terms of pixel intensities, and re-expresses them in terms of frequencies, storing coefficients losslessly; these techniques are commonly used with, e.g., JPEGs (Anderson, 1996; Zhao and Koch, 1995; Ó Ruanaidh et al., 1996). Many existing steganographic systems make use of DCT coefficients, including the *F5* (Westfeld, 2001) and *Outguess* (Provos, 2001) algorithms, together with other *model-based* (Sallee, 2003, 2005), *modified matrix* (Kim et al., 2006) and *perturbed quantization* methods (Fridrich et al., 2005).

Of more significance for our purposes, Prabhakaran and Shanthi (2012) describe a hybrid crypto-steganography method, which Shanableh (2012) extends by encoding data in the motion vector and quantisation scales (section 1.5); their technique increases the steganographic capacity of the file, but is generally limited to raw video. Fang and Chang (2006) focus on modifying the motion vectors of fast-moving objects (since such changes are relatively undetectable). In contrast, Aly (2011) examines macroblocks to determine which are most suitable for LSB embedding; both papers report that the resultant video quality remains good, but this is not quantified.

1.5 Video Encoding

In this section we briefly describe the structure of a typical video file. Different types of video frame serve different purposes, and it is essential for steganographic purposes that only certain kinds of frame are manipulated; attempting to hide data within the wrong frames typically causes the covert message to become garbled during re-extraction. We consider these practical issues in more detail in section 2 below.

1.5.1 Coding Concepts

An encoder (compressor) and decoder (decompressor) forming a complementary pair is known as a *codec* (encoder/decoder). The encoder is used to store or transmit video by converting the original raw video format to an alternative (compressed) representation. The decoder converts the compressed form back to the original video.

In general terms, a video encoder implements three main components: a *temporal model*, a *spatial model* and an *entropy encoder*. The temporal model reads in a sequence of video frames and attempts to reduce redundancy by identifying similarities between neighbouring frames – this analysis usually involves computing a prediction of the current video frame. With H.264 the prediction can be computed from multiple previous or future frames. The prediction is improved by means of compensation for differences between the frames – this is known as motion compensation prediction. The temporal model outputs a residual frame and a set of *motion vectors*. The residual frame is computed by subtracting the prediction from the current frame, and motion vectors are used to describe how the motion was compensated.

The residual frame from the temporal model is then fed into the spatial model. This step is again concerned with removing redundancy, but in this case by analysing neighbouring samples within the frame itself. Spatial reduction in H.264 is achieved by applying a transform followed by a *quantisation* process. Quantisation is the process of scaling down the range of symbols that are used in a representation. For instance, the DCT transform produces a matrix of coefficients whose values may range between -223 and 150 , but after quantisation, these values may only range between 10 and 130 . This reduced range means that fewer bits are needed to code the representation than the original range, and this can lead to significant (but lossy) compression (section 1.6). Quantisation parameters for multimedia formats are chosen based on how individual components affect the average human perception (Mukhopadhyay, 2011). The transform step produces a set of transform coefficients which are then quantised, removing insignificant values, and returning the quantised transform coefficients as the output of the spatial model.

Finally, the entropy encoder produces an encoded output from the results of the spatial and temporal models. It processes the motion vectors from the temporal model and the coefficients from the spatial model to produce a compressed bit stream consisting of motion vectors, residual transform coefficients and header information.

Although the quantisation stage causes a loss of information, this process is roughly reversible, and the decoder mechanism essentially ‘works in reverse’ to retrieve the original video. Nonetheless, the output produced by the decoder mechanism will only ever (in the case of H.264) be an approximation to the original input because of the quantisation stages.

1.5.2 Temporal Model

The residual frame produced by the temporal model is produced by subtracting the predicted frame from the actual video frame, and its size is dependent on the accuracy of the prediction process – the smaller the residual frame, the fewer bits needed to code it. Prediction accuracy can be improved by calculating and propagating compensation for motion from the reference frame(s) through to the current frame.

Motion compensation can significantly improve prediction calculations because two successive video frames are usually highly correlated, because most of the information captured in successive residual frames relates to the movement of objects in an essentially static scene. These changes directly correspond to the movement of pixels between frames, a feature known as *optical flow* (Ahmad et al., 2005).

In theory, knowing the optical flow allows us to predict the majority of the pixels in the current frame, simply by displacing pixels in the preceding frame as required. Unfortunately, this is a very computationally intensive process, as each pixel will have to be transformed, and each frame decoded, on a pixel-by-pixel basis using the optical flow vectors. Whilst workable in theory, this would result in a large amount of residual data, which is at odds with the desirability of a compact residual frame.

1.5.3 Macroblock Motion Estimation

A macroblock is typically a 16×16 pixel block of the current frame, although in the wider context of block-based motion estimation other suitably-sized $N \times M$ samples might be used. Macroblocks are used by a variety of codecs including MPEG-1, MPEG-2, H.261, H.263 and H.264.

Macroblock motion estimation starts by dividing frames into macroblocks. Each macroblock is taken in turn, and a previously selected ‘reference frame’ is searched for a matching macroblock. Macroblocks from the reference frame are paired with macroblocks in the current frame by choosing a candidate block that minimises the difference between the macroblock in the current frame and itself – this process provides a *residual block*. Finally, the residual block is encoded and stored, together with the associated motion vector.

Using a 16×16 size macroblock can cause some problems with certain motions and object outlines. If a macroblock and its matching macroblock differ greatly, the number of bits required for the encoding increases and inflates the bit-rate. This issue can be addressed by decomposing a macroblock into smaller 8×8 , or even 4×4 macroblock size, but this results in a larger number of blocks, which can be disadvantageous. The H.264 codec overcomes this problem to some extent by adopting an ‘adaptive’ block size approach.

1.6 Compression

An image in a video stream can be thought of as a function which maps each point of a 2D spatial domain to a three-dimensional RGB colour vector. Consequently, if we were to store a single 1920×1080 image in raw format, just over 2 million RGB triples would need to be stored. This is a substantial amount of information to store for a single image. Given that videos typically run at between 24 and 30 frames per second, a single second of video footage at 1920×1080 resolution would require the storage of around 50–60 million RGB triples. Storing and/or streaming this data in a raw, uncompressed format is consequently

impractical for most situations, and as a result image and video formats typically use an alternative, compressed, representation.

1.6.1 Compression Considerations

Data compression inevitably involves trade-offs between computational costs, storage requirements, and accuracy of representation. For images and video, an approximation of the original source is often sufficient, which means that lossy compression schemes can be used, but if the given application requires complete accuracy then a lossless representation must be used.

In section 1.4 we saw how coefficient-based transforms such as DCT can be used to represent an image, and these techniques can be used to assist the image compression process. However, simply compressing each frame in turn is inefficient, since it fails to take into account the temporal cohesion between consecutive frames. Examining the correlation between consecutive frames typically allows a more concise representation to be used (Mukhopadhyay, 2011; Richardson, 2008; Le Gall, 1991).

A common approach is to use a GOP (“group of pictures”) structure. In a GOP, a reference frame is chosen, which is called an *Intra Frame* or *I-Frame*. Other frames are then predicted from this, where predictions are represented as changes (deltas) from the preceding frame – these are known as *P-Frames*. Some frames are predicted using both the preceding and subsequent neighbours, and these are known as bi-directionally predicted frames or *B-Frames* (Mukhopadhyay, 2011; Richardson, 2008).

2 Development Issues

Given the experimental focus of this work, we adopted an ‘agile’ methodology and developed various auxiliary tools for embedding messages in audio and image files. In order to ensure usability, we designed the system as a user-friendly GUI, interacting with a lower-level suite of tools housing the core steganographic logic. We had intended using the same programming language throughout, but this proved infeasible, so different parts of the system had to be constructed using different programming languages.

In particular, there was insufficient time to develop a complete video coding tool from scratch, so we adopted third-party libraries. We began with Xuggler,¹ a Java wrapper for FFmpeg,² but unfortunately Xuggler proved insufficiently flexible, and we found it necessary to work directly with the FFmpeg library using code written in C. Even so we needed to re-implement part of the FFmpeg library – we are grateful to Michael Niedermayer, one of FFmpeg’s developers, for vital feedback at this time (personal communication).

Since we were already using C for the low-level video coding tools, we considered using it for the GUI as well, but this would have required using the GTK+ library,³ which has poor Mac OS integration. Moreover, good GUIs should be multithreaded to ensure the display updates regularly, but C has no uniform cross-platform API for managing threads. We therefore re-adopted Java and Xuggler for GUI development (Java includes GUI design packages, while Xuggler can be used for video playback).

¹<http://www.xuggle.com/xuggler>

²<http://ffmpeg.org/>

³<http://www.gtk.org/>

2.1 Choice of codec

For development purposes we limited our choice of codecs to those supported by FFmpeg, focussing eventually on ‘H.264’, since this is the most common video codec used by modern cameras (Case, 2010) and for online videos (Schonfeld, May 1st, 2010).

2.2 Transcode Mechanism

Video transcoding involves demultiplexing the original input file to distinguish audio from video data. The separate streams are then processed independently and re-encoded back into the output file. Our transcoder uses FFmpeg’s `avcodec` and `avformat` libraries. The input is scanned for audio and video streams, and relevant codecs are loaded into memory. A header is written to the output file, and we iterate over input data packets. Finally, a suitable footer is appended.

Though simple in theoretical terms, decoding input packets proved somewhat problematic in practice. Frames produced by FFmpeg’s decoder methods `avcodec_decode_video2` and `avcodec_decode_audio4` cannot be parsed directly without preparation, because they lack appropriate timestamps.⁴ Failure to set these timestamps results in either no video image, or else a lack of synchronisation between audio and video streams. Frames decoded by `avcodec_decode_video2` also contained data that interfered with the encoding, causing the image to be heavily pixelated. We eventually solved this problem by copying raw image data to a new `AVFrame` instance, which allowed us to carry out the requisite pre-parsing preparation.

2.2.1 Data encoding/decoding

Data encoding and decoding was performed by modifying motion vectors using a callback (figure 2). This approach makes it easy to implement additional steganographic schemes: the motion vector and frame number are passed to our bespoke callback method, `stegEncodeMv` (invoked from inside the `avcodec` library), which chains a callback to the relevant encoder modules, as determined by the current `getStegEncoderMode` (figure 3). The decoding process is somewhat simpler – it uses a single `Decoder` component (figure 3(b)), iterating the relevant decoding method over the packets of the specified video file.

2.3 Steganographic encoding/decoding

We developed a process to modify the `AVFrame` parsed to the `avcodec_encode_video2` method for coding into a compressed packet, but despite our best efforts the alterations made to `motion_val` were not reflected in the output. After dissecting the 850,000+ lines of code comprising the FFmpeg codebase, we deduced that adjusting the behaviour of `ff_estimate_p_frame_motion` in `libavcodec/motion_est.c` would let us manipulate the motion vectors of macroblocks in P-Frames.

Initially, our encoder modified vectors using a bit mask, but tests showed that only around 30 characters could be embedded, and roughly 50% of embedded bits would ‘flip’, causing inaccurate decoding of the hidden message. Using different bit masks provided little improvement.

⁴A timestamp mechanism is used to synchronise different streams in a video file.

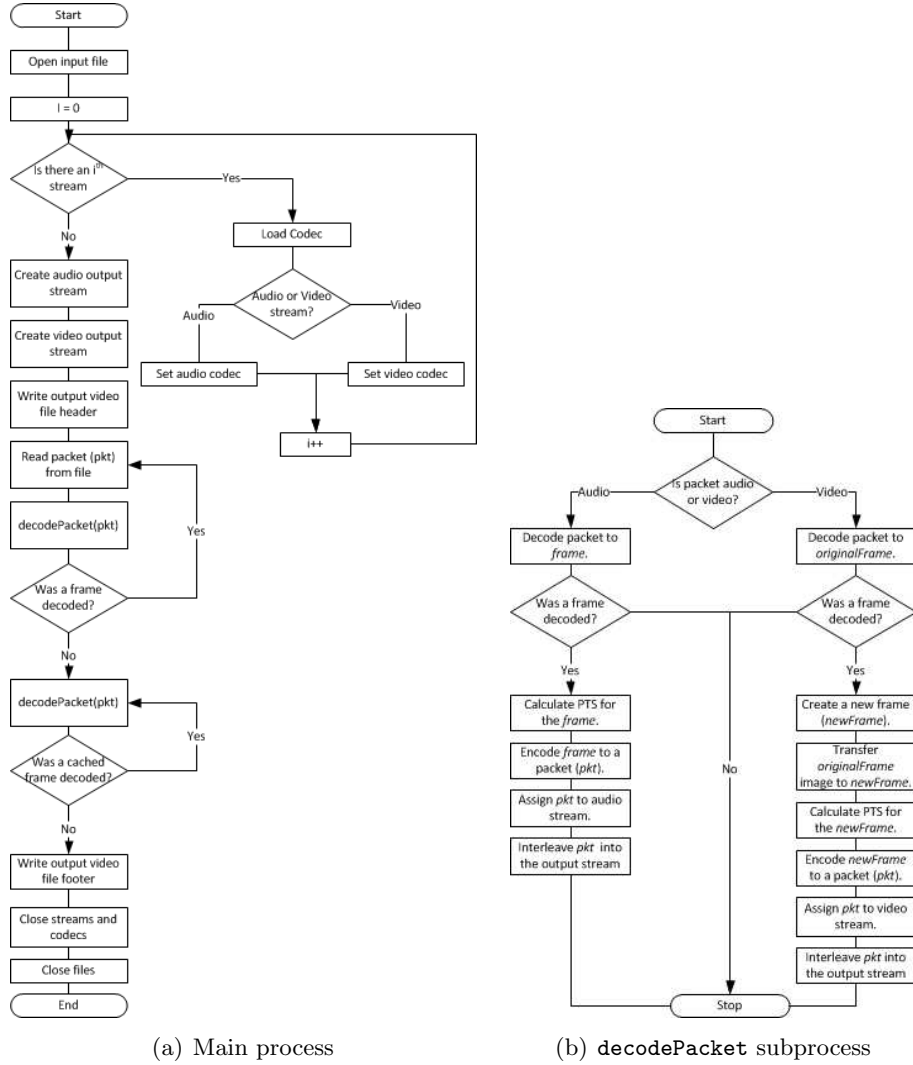


Figure 2: Transcode process flow charts

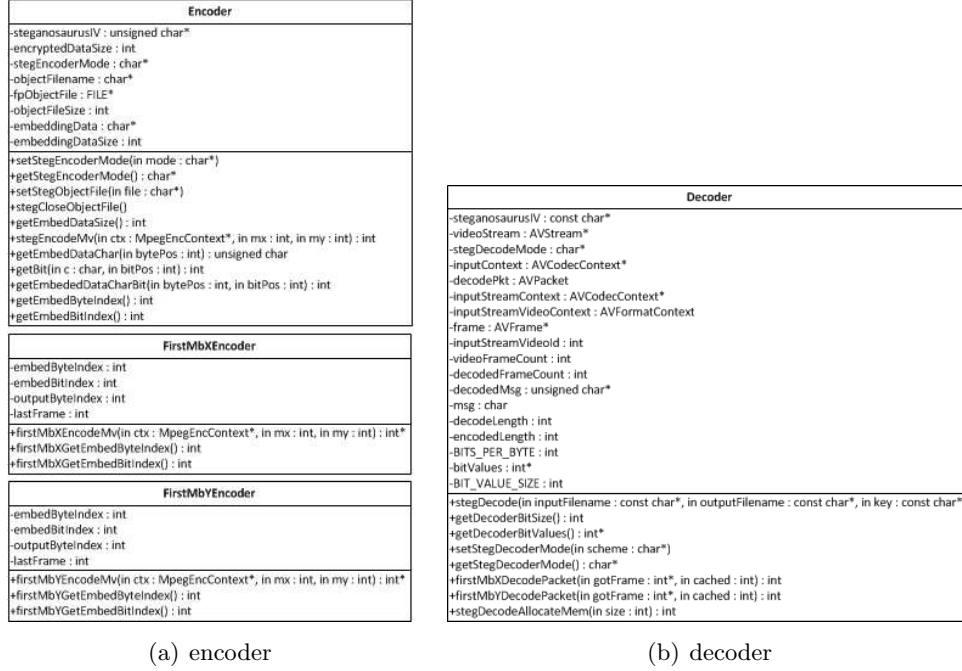


Figure 3: Encoder and decoder architecture overviews

Eventually, we discovered that implementing our callback in the `ff_estimate_p_frame_motion` method of `libavcodec/motion_est.c` caused data to be embedded prior to quantisation, on occasion obliterating our changes. We therefore moved our callback to the `encode_mb_internal` method of `libavcodec/mpegvideo_enc.c`. Further analysis revealed that certain values returned by the decoder varied seemingly at random. We deduced that the vector coding process performs an additional right-shift operation, and therefore introduced a shifted encoding mask. The problem was fully resolved when we discovered that certain macroblocks were flagged as having no motion vector. After moving the encoder callback, compensating for bit shifts and avoiding vector-less frames our transcoding mechanism was finally capable of embedding data in video.

The decoder is far simpler, and operates by parsing AVPackets to `avcodec_decode_video2` until a complete AVFrame has been returned.

2.4 Cryptography Subsystem

It was not feasible, given the lifespan of the project, to determine whether our system exhibits longterm security. We therefore incorporated well-established cryptographic methods to ensure a minimum level of communications security. Our system fully implements the AES cryptosystem (Cid et al., 2006), and comprehensive unit testing was employed to ensure that our implementation conforms to the relevant FIPS-197 specification (National Institute of Science and Technology, 2001). While encryption and steganography serve fundamentally different purposes, encryption can nonetheless be used *with* steganography to obfuscate embedded ASCII messages: figure 4(b) shows a significant increase in frequency for key ASCII values (especially spaces and alphabetic characters), but this tell-tale signature is removed by encrypting prior to embedding (figure 4(c)).

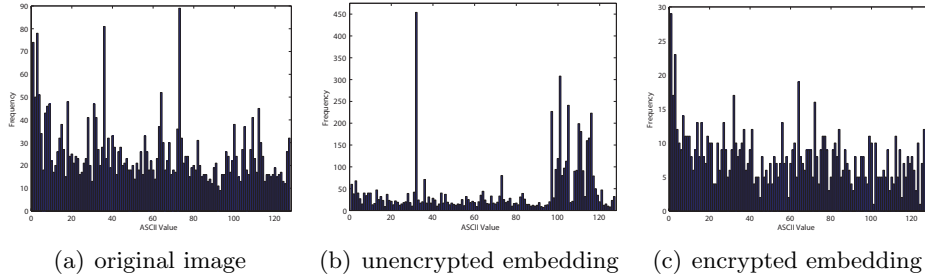


Figure 4: ASCII distributions from the LSB string of a PNG image (a) before and (b) after unencrypted data is embedded; (c) the obfuscating effect of encrypting data prior to embedding.

3 Evaluation and Conclusion

Our initial intention was to develop numerous schemes for embedding and extracting data within video streams. This was overly ambitious, as we significantly underestimated the complexities associated with manipulating video. We nonetheless managed to develop two different methods for embedding and re-extracting data from video. Our final solution uses motion vector based approaches.

This project also proved substantially more experimental than initially predicted. Our initial literature survey and preliminary research provided surprisingly little insight into how to design and implement a practical steganographic system. We therefore adopted an agile philosophy, and as the project progressed numerous design and implementation changes were undertaken and important lessons were learned. Although time constraints prevented us exploring more detailed steganographic schemes, our tools can take an object file in any format, apply AES encryption, and embed the data so that the resultant video is indistinguishable from the original and is capable of normal video playback. As part of testing and evaluation we made binary executables publicly available⁵.

3.1 Technical limitations

Our main goal was to develop and research techniques for embedding data using motion vector based techniques. While this has ultimately proven successful, we encountered notable setbacks. We thought it would be possible to embed data in motion vectors of any P- or B-frame, but discovered in some cases that a macroblock can be coded as having *no* motion vector (as opposed to one of zero magnitude). This is an important distinction that needs to be considered carefully when choosing the specific frames in which to embed covert data.

Another limitation is our inability to determine steganographic capacity in advance. Motion vectors describe the spatial translation of a pixel block between frames, whence modifying one frame will impact its neighbours. Moreover, the number of encodable macroblocks changes with the object to be embedded, and can also vary due to keyframe positions or GOP sizes. This sometimes changes B- or P- frames to I-frames which have no macroblocks, dramatically changing the number available. Whilst we could change our system to preserve I-frame positions, we found that regular GOP sizes provide better error correction.

⁵<http://steganosaur.us/download>

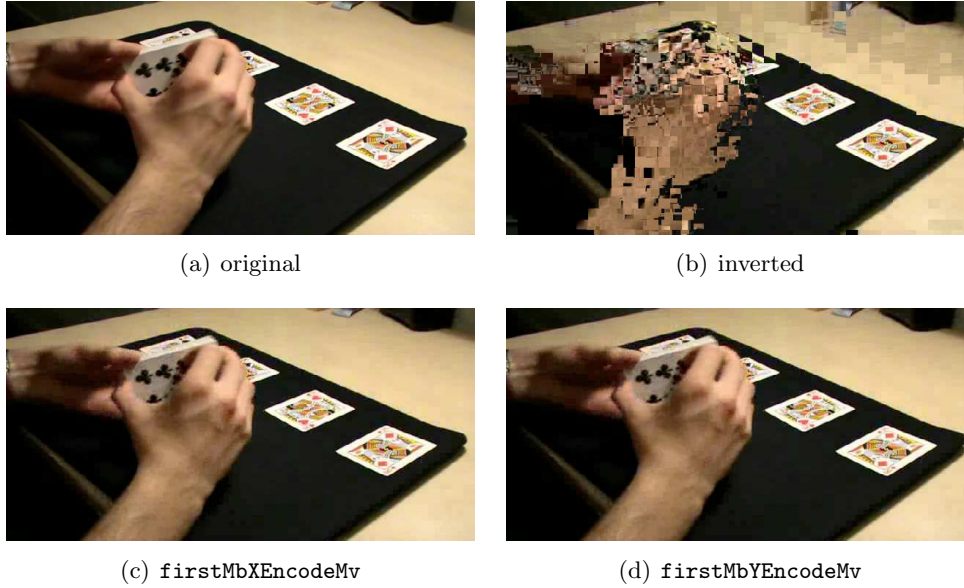


Figure 5: The visible effects of various encodings: (a) original video frame; (b) inverted motion videos producing visible artefacts; (c, d) using our two encoding functions produces no obvious artefacts.

Nonetheless, we have produced a system that achieves most of the goals we set (see A for examples of some of our outputs). The project was considerably more complex than originally envisaged, but we have largely been able to identify and overcome the major hurdles we encountered.

A Typical system outputs

Figure 5 shows an unmodified frame (a) from one of our test videos, together with the effects of applying various modifications. In (b) we see how inappropriate encoding produces identifiable distortion. However, user testing showed that neither of the bespoke encoding methods `firstMbXEncodeMv` (c) or `firstMbYEncodeMv` (d) produced visibly identifiable artefacts. Figure 6 shows the corresponding motion vectors in the original and `firstMbXEncodeMv` versions of a frame. The X-component of this frame was not modified, but there is nonetheless significant fluctuation in the motion vector values, due to the lossy nature of H.264/MPEG4 coding.

References

- Ahmad, I., Wei, X., Sun, Y., Zhang, Y., 2005. Video Transcoding: An Overview of Various Techniques and Research Issues. *IEEE Transactions on Multimedia* 7, 793–804.
- Al-Frajat, A.K., Jalab, H.A., Kasirun, Z.M., Zaiden, A.A., Zaiden, B.B., 2010. Hiding Data in Video File: An Overview. *Journal of Applied Sciences* 10, 1644–1649.

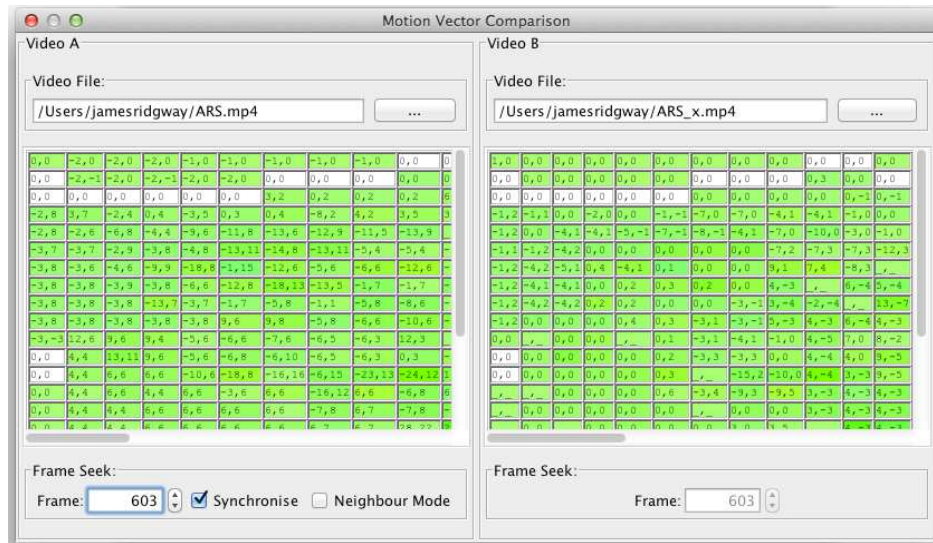


Figure 6: Motion vector comparison of figures 5(a) and 5(c).

- Aly, H.A., 2011. Data Hiding in Motion Vectors of Compressed Video Based on Their Associated Prediction Error. *Information Forensics and Security, IEEE Transactions on* 6, 14–18. doi:10.1109/TIFS.2010.2090520.
- Anderson, R., 1996. Stretching the Limits of Steganography. *IEEE Journal of Selected Areas in Communications* 16, 474–481.
- Case, L., 2010. All About Video Codecs and Containers. http://www.pcworld.com/article/213612/all_about_video_codecs_and_containers.html. Date Accessed: 9 April 2013.
- Cid, C., Murphy, S., Robshaw, M., 2006. *Algebraic Aspects of the Advanced Encryption Standard*. Springer, New York.
- Cole, E., 2003. *Hiding in Plain Sight: Steganography and the Art of Covert Communication*. Wiley Publishing, Inc.
- Eltahir, M.E., Kiah, L.M., Zaidan, B.B., Zaidan, A.A., 2009. High Rate Video Streaming Steganography, in: *Proceedings of the 2009 International Conference on Future Computer and Communication*, IEEE Computer Society, Washington, DC, USA. pp. 672–675. URL: <http://dx.doi.org/10.1109/ICFCC.2009.44>.
- Fang, D., Chang, L., 2006. Data hiding for digital video with phase of motion vector, in: *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pp. 1422–1425. doi:10.1109/ISCAS.2006.1692862.
- Fridrich, J., 2010. *Steganography in Digital Media: Principles, Algorithms and Applications*. Cambridge University Press.
- Fridrich, J., Goljan, M., Soukal, D., 2005. Perturbed quantization steganography. *Multimedia Systems* 11, 98–107.

- IBM, Microsoft, 2004. Multimedia Programming Interface and Data Specifications 1.0. <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf>. URL: <http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/Docs/riffmci.pdf>. accessed: 16 June 2013.
- Jalab, H.A., Zaidan, A.A., Zaidan, B.B., 2009. Frame Selected Approach for Hiding Data within MPEG Video Using Bit Plane Complexity Segmentation. *Journal of Computing* 1, 108–113.
- Johnson, N.F., Duric, Z., Jajodia, S., 2003. *Information Hiding: Steganography and Watermarking - Attacks and Countermeasures (Advances in Information Security)*. Kluwer Academic Publishers.
- Kim, Y., Duric, Z., Richards, D., 2006. Modified Matrix Encoding Technique for Minimal Distortion Steganography, in: *Information Hiding*, Springer. pp. 314–327.
- Le Gall, D., 1991. MPEG: a video compression standard for multimedia applications. *Commun. ACM* 34, 46–58.
- Marincola, J.M., 1996. *Herodotus: The Histories*. Penguin Books.
- Mukhopadhyay, J., 2011. *Image and Video Processing in the Compressed Domain*. CRC Press.
- National Institute of Science and Technology, 2001. Federal Information Processing Standards Publication 197. <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. URL: <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>. accessed: 9 April 2013.
- Noda, H., Furuta, T., Niimi, M., Kawaguchi, E., 2004. Application of BPCS steganography to wavelet compressed video, in: *Image Processing, 2004. ICIP '04. 2004 International Conference on*, pp. 2147–2150.
- Ó Ruanaidh, J.J.K., Dowling, W.J., Boland, F.M., 1996. Watermarking digital images for copyright protection. *Vision, Image and Signal Processing, IEE Proceedings* 143, 250–256.
- Prabhakaran, B., Shanthi, D., 2012. A New Cryptic Steganographic Approach using Video Steganography. *International Journal of Computer Applications* 49, 19–23.
- Provos, N., 2001. Defending Against Statistical Steganalysis, in: *10th USENIX Security Symposium*, pp. 323–335.
- Richardson, I.E.G., 2008. *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley & Sons.
- Ridgway, J., 2013. *Video Steganography*. Project dissertation. Department of Computer Science. University of Sheffield, UK.
- Sallee, P., 2003. Model-Based Steganography, in: *International Workshop on Digital Watermarking*, Springer. pp. 154–167.
- Sallee, P., 2005. Model-Based Methods for Steganography and Steganalysis. *International Journal of Image and Graphics* 5, 167–189.

- Schonfeld, E., May 1st, 2010. H.264 Already Won—Makes Up 66 Percent Of Web Videos. <http://techcrunch.com/2010/05/01/h-264-66-percent-web-video/>. Date Accessed: 9 April 2013.
- Shanableh, T., 2012. Matrix encoding for data hiding using multilayer video coding and transcoding solutions. *Signal Processing: Image Communication* 27.
- Singh, S., Agarwal, G., 2010. Hiding image to video: A new approach of LSB replacement. *Int. J. Engineering Science and Technology* 2, 6999–7003.
- Westfeld, A., 2001. F5 – a steganographic algorithm: High capacity despite better steganalysis, in: 4th International Workshop on Information Hiding, Springer-Verlag. pp. 289–302.
- Zhao, J., Koch, E., 1995. Embedding Robust Labels into Images for Copyright Protection, in: Brunnstein, K., Sint, P.P. (Eds.), *Intellectual Property Rights and New Technologies, Proceedings of the KnowRight 95 Conference, 21.-25.8.1995, Wien, Austria*, Austrian Computer Society. pp. 242–251.